Methods and Experiments With Bounded Tree-width Markov Networks

Percy Liang, Nathan Srebro {PLIANG,NATI}@MIT.EDU
MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA 02139, USA

Abstract

Markov trees generalize naturally to bounded tree-width Markov networks, on which exact computations can still be done efficiently. However, learning the maximum likelihood Markov network with tree-width greater than 1 is NP-hard, so we discuss a few algorithms for approximating the optimal Markov network. We present a set of methods for training a density estimator. Each method is specified by three arguments: tree-width, model scoring metric (maximum likelihood or minimum description length), and model representation (using one joint distribution or several class-conditional distributions). On these methods, we give empirical results on density estimation and classification tasks and explore the implications of these arguments.

1. Introduction

Density estimation is a useful tool that can be applied to a variety of tasks involving inference, classification, and prediction. The problem is to train a model involving a set of interdependent variables given data points drawn from some fixed distribution. Markov networks use undirected graphs to explicitly represent the dependencies (Pearl, 1997). The qualitative graph structure gives an intuitive explanation of the model for humans while the quantitative parameters provide the model with rigor.

In general, learning the best model with respect to a metric such as Bayesian score is NP-hard (Chickering et al., 1994), but the problem is tractable if we restrict our family of models. For instance, Chow and Liu (1968) showed that the maximum likelihood Markov tree can found exactly in polynomial time. Their work has spurred many generalizations that permit more dependencies: mixture trees (Meila & Jordan, 2000), thin junction trees (Bach & Jordan, 2001), polytrees (Dasgupta, 1999), and large node Chow-Liu trees (Huang et al., 2002a).

In this paper, we focus on one specific generalization of Chow and Liu (1968)'s work: bounded tree-width Markov networks (Srebro, 2000). Learning the maximum likelihood bounded tree-width Markov network reduces to the problem of finding the maximum weight hyperforest in a hypergraph, which is NP-hard even for tree-width 2. Section 3 discusses various algorithms for approximating the maximum hyperforest.

The maximum likelihood (ML) score is not suitable for model selection because it will always add the maximum number of hyperedges to the model. We consider using an alternative scoring metric based on minimum description length (MDL) (Rissanen, 1987) for regularization.

If all variables are treated equally, then we would train a single joint model over all the variables. But if there is a class variable, it might be advantageous to train a separate model for each class. In that case, we can either force each class-conditional model to have a single shared graph structure or let them have varying structures. We consider all three of these model representations.

Thus, there are three arguments that specify a method for training a density estimator: tree-width $(0, 1, 2, \ldots)$, scoring metric (ML or MDL), and model representation (joint, conditional with shared structure, or conditional with free structure). We present the first set of comprehensive experimental results on a large number of empirical data sets for each of these methods (Section 6).

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

2. Bounded tree-width Markov networks

2.1. Overview

This section reviews bounded tree-width Markov networks (Srebro, 2000). A Markov network associated with a graph G specifies a probability distribution P_G. The n vertices (variables) of G are connected by edges, which represent allowable dependencies between variables. A variable v, conditioned on its neighbors, is independent of all other variables.

The distribution P_G of the Markov network can be decomposed into the product of potential functions over the cliques[1] in G:

$$P_G(\mathbf{x}) = \prod_{h \in \text{Cliques}(G)} \phi_h(\mathbf{x}_h) \tag{1}$$

A word about notation: if $\mathbf{x}$ is a vector of the values of all the variables, then $\mathbf{x}_h$ is the vector of the values of the variables in h. If G is triangulated[2], exact inference and computation of marginal probabilities are possible. If the maximum clique size $k+1$ of G is small, then relatively few parameters are necessary to specify P_G. If these two properties are satisfied, then G has *tree-width* k.

Every tree-width k graph G has a covering k-*hyperforest* H whose hyperedges are exactly the cliques (not necessarily maximal) in G. From now on, we will speak of hyperedges in H rather than cliques in G.

In that case, each potential function ϕ_h can be expressed purely in terms of the marginal probabilities over $\mathbf{x}_h$, completely independent of the structure and marginals elsewhere in the hyperforest in the following way:

$$\phi_h(\mathbf{x}_h) = \frac{P_h(\mathbf{x}_h)}{\prod_{h' \subsetneq h} \phi_{h'}(\mathbf{x}_{h'})} \tag{2}$$

The parameters that specify the Markov network $P_G = P_H$ are the marginal probabilities P_h over the hyperedges $h \in H$. If the hyperforest H is fixed, and our data points are drawn independently from some fixed target distribution $\hat{P}$. The maximum likelihood Markov network P_H over H is the one in which $P_h = \hat{P}_h$ for all hyperedges $h \in H$.

In the following two sections, we show how the two scoring metrics, ML and MDL, can be expressed as a sum of weights over the hyperedges of a hyperforest (times a constant). Then to find the best k-hyperforest, we can compute in advance all the weights w_h of the candidate ($\leq k$)-hyperedges, and feed these abstracted weights into an algorithm for finding the maximum weight hyperforest. It is crucial that the computation of w_h depend only on $\hat{P}_h$.

Finding the best bounded tree-width Markov network with respect to a scoring metric reduces to finding the maximum weight hyperforest with the corresponding weights. Section 3 will discuss algorithms for finding the maximum weight hyperforest.

2.2. Weights for likelihood

The likelihood of P_H with respect to $\hat{P}$ is given by the following:

$$\begin{aligned} L(H) &= -D(\hat{P} \parallel P_H) \\ &= E_{X \sim \hat{P}}\big[\log P_H(X)\big] \\ &= E_{X \sim \hat{P}}\big[\log \prod_{h \in H} \phi_h(X_h)\big] \\ &= \sum_{h \in H} E_{X_h \sim \hat{P}_h}\big[\log \phi_h(X_h)\big] \\ &\triangleq \sum_{h \in H} w_h \end{aligned}$$

In the last step, we defined the weight w_h of an hyperedge h to be a function of the target marginal probabilities over h. If h is a single vertex v, then w_h is $-H(P_v)$, the negative entropy of the marginal probability distribution with respect to variable v. If h is an edge $\{u, v\}$, then w_h is $I(X_u; X_v) = H(X_u) + H(X_v) - H(X_u, X_v)$, the mutual information between random variables X_u and X_v.

All weights except those of single vertices are *monotone*, meaning that the weight of any hyperforest H is at least the weight of any sub-hyperforest $H' \subset H$. The single vertices always have negative weight and are included in any hyperforest by default.

2.3. Weights for minimum description length

Since the weights for the likelihood scoring metric are monotone, the maximum weight hyperforest will always be a hypertree. However, the hypertree may contain more hyperedges than warranted, and is bound to overfit the data, even though we are already limiting the tree-width.

To regularize the model for model selection, we can find the bound tree-width Markov network with the minimum description length (MDL) (Rissanen, 1987; Bouckaert, 1994). The description length of P_H with

[1] A clique is a fully-connected subgraph.

[2] A graph is triangulated if there are no minimal cycles of more than 3 edges.

respect to m data points drawn from $\hat{P}$ is as follows:

$$\begin{aligned} DL(H) &= mD(\hat{P} \parallel P_H) + \frac{1}{2} N_H \log m \\ &= mD(\hat{P} \parallel P_H) + \frac{1}{2} \log m \sum_{h \in H} N_h \\ &= -m\Big(\sum_{h \in H} w_h - \sum_{h \in H} \frac{N_h}{2m} \log m \Big) \\ &= -m \sum_{h \in H} \Big(w_h - \frac{N_h}{2m} \log m \Big) \\ &\triangleq -m \sum_{h \in H} w'_h \end{aligned}$$

N_H is the number of parameters in H, which can be decomposed into N_h over all hyperedges $h \in H$. Let $N_h = \text{NumParams}(P_h) - \sum_{h' \subsetneq h} N_{h'}$.

$\text{NumParams}(P_h)$ is the number of parameters needed to specify the marginal probabilities of P_h. If h contains d discrete variables, and the variables can take on $n_1, n_2, \ldots, n_d$ different values, respectively, then $\text{NumParams}(P_h) = \big(\prod_{i=1}^{d} n_i \big) - 1$.

Clearly, minimizing the description length is equivalent to maximizing the weight of the hyperforest. Note that these weights w'_h are no longer monotone, since we have introduced a penalty on each hyperedge that is related to the number of parameters associated with it.

3. Maximum hyperforest algorithms

Now we turn our attention to finding the maximum weight hyperforest in a hypergraph. For tree-width 1, the problem is essentially the standard maximum spanning tree problem (Cormen et al., 1989). Note that if we use MDL, we might have negative weights on some edges, in which case we might end up with a forest instead of a tree.

The two most common algorithms for finding the maximum spanning tree are due to Kruskal and Prim. The former takes a global approach by greedily adding the maximum weight edge that does not form a cycle. The latter takes an incremental approach by starting at a vertex and greedily connecting new vertices to the current tree to maximize the weight of the resulting tree.

3.1. A global algorithm

Srebro (2000) presents a randomized approximation algorithm that finds a hypertree whose weight is at least $1/(8^k k!(k+1)!)$ times the weight of the optimum hypertree (assuming weights are monotone). The algorithm first approximates the maximum weight *windmill farm* in the hypergraph. A windmill farm is a special kind of hyperforest, so it can greedily extended by adding hyperedges, as long as we maintain acyclicity. To facilitate the greedy Kruskal-like extension, (Liang & Srebro, 2003) developed a data structure for detecting hypercycles.

We can consider a simpler algorithm that does the greedy extension starting with an empty hyperforest instead of a windmill farm. This algorithm has no theoretical guarantees. But we tested these two variants on both artificial and real data and found that in practice, the algorithm based on windmill farms performs no better. This is not too surprising given the weak lower bound.

3.2. An incremental algorithm

Instead of taking a global approach to the problem, consider incrementally constructing a hyperforest[3] from some initial hyperedge (Malvestuto, 1991). At each iteration of the algorithm, we choose a new vertex v to connect to the current hyperforest via a hyperedge h, as to maximize the weight of the resulting hyperforest. h must be the union of v and some (not necessarily maximal) hyperedge h' in the current hyperforest. Note that if h' is the empty hyperedge, the new hyperedge h would just be $\{v\}$, which breaks away from the current hyperforest. When h is added, all sub-hyperedges of h not already in the hyperforest must also be added.

The incremental algorithm has the computational advantage that we do not need to detect hypercycles, which is a complicated task. The algorithm always maintains a hyperforest, since it is adding hyperedges in reverse order of a Graham reduction.

One might expect that the global algorithm to perform better than the incremental algorithm because it makes choices that are in some sense more globally wise. But surprisingly, based on empirical evidence, exactly the opposite is true.

Finally, we can incorporate limited backtracking into these greedy algorithms to increase quality. Our backtracking variant of the incremental algorithm tries each of the 100 heaviest hyperedges as the starting hyperedge. It chooses the best resulting hyperforest. This simple modification increased the hyperforest weight a modest amount. This is the variant that we will use from now on.

[3]This hyperforest will always be a hypertree if all weights are monotone.

4. Model representation and classification

If we are simply want a density estimator and no variable is distinguished as a class variable, then we can find a joint distribution to model the data using the techniques discussed in Section 3. However, in classification tasks, the class variable is distinguished. It might be beneficial to model each class separately and then combine all the class-conditional models into our final model. For notational purposes, let us split a data point $\mathbf{x}$ into the class $y \in Y$ and the values of all other variables $\mathbf{z}$. The three model representations are detailed below.

Joint (J) We model the data directly using a single Markov network trained on all our training data: $P(\mathbf{x}) = P(y, \mathbf{z})$.

Class-conditional with free structure (C_f) We build a separate model $P_y(\mathbf{z})$ for each class y using only the data with the corresponding class. Our final model is described by $P(\mathbf{x}) = P(y, \mathbf{z}) = P(y)P(\mathbf{z}|y) = P(y)P_y(\mathbf{z})$. Each Markov network $P_y(\mathbf{z})$ may have a different structure.

Class-conditional with shared structure (C_s) Like C_f, we build $P_y(\mathbf{z})$ for each class, but now, we want to enforce that they all have the same structure. To accomplish that goal, for each class y, we compute the vector of weights $\mathbf{w}_y$ over all hyperedges not involving y. We find the best shared hyperforest using a convex combination of these weights: $\mathbf{w} = \sum_{y \in Y} P(y)\mathbf{w}_y$. C_s is a compromise between J and C_f in that the parameters are specific to each class, but the structure is global.

We can directly use a probabilistic model as a classifier by doing inference on the missing class variable. Given input $\mathbf{z}$, we output $\text{argmax}_y P(y, \mathbf{z})$. Such a classifier has the advantage that if other variables in $\mathbf{z}$ have missing values, they can be inferred using the same machinery.

Note that for the J representation, only the vertices directly connected to the class variable are relevant for classification.[4]

5. Related work

Each method shall be denoted by its three arguments as f-R-k, where k is the tree-width, R is the model representation, and f is the scoring metric. Notice that changing an argument affects the complexity (number of parameters) of the model: higher tree-width results in more complex models than lower tree-width; ML is more complex than MDL; C_f is more complex than C_s which is more complex than J. The simplest model is MDL-0-J, and the most complex model is ML-3-C_f.

{MDL,ML}-0-J	Choose the most common label
{MDL,ML}-{C_f,C_s}-0	Naive Bayes (NB)
ML-C_f-1	Chow and Liu (CL)
ML-C_s-1	Tree-Augmented Naive Bayes (TAN)

Table 1. Relating previous work to our methods.

Some of these methods are old news. Three methods that have been Table 1 shows how those methods fit into our framework.

Friedman et al. (1997) discusses the three non-trivial methods in Table 1 as well as a algorithm that builds a Bayesian networks by adding edges based on MDL score. An issue with the algorithm is that it would stop adding edges if variables were pairwise independent, but there are dependencies involving more than two variables. Our algorithm considers all k-hyperedges at once, so we can capture up to $(k+1)$-th order dependencies. Of course, we could still fail if we were trying to learn the parity function over all n variables, in which case any proper subset of the n variables exhibit independence, but the n variables are dependent.

Huang et al. (2002a) builds a classifier from ML-k-C_f. To construct the hyperforest, they first finds the maximum likelihood Markov tree and then greedily contracts edges into *large nodes* of maximum size j. thus constructing a $(2j-1)$-hyperforest. In their experiments, they considered $j = 3$.

Semi-naive Bayes models are hyperforests in which all the hyperedges intersect at exactly the class variable. They form a subset of ML-k-C_s. Huang et al. (2002b) uses uses linear programming relaxation to approximate the maximum likelihood model.

6. Experiments

6.1. Synthetic data: recovering a hidden hypertree

We generated a random tree-width k Markov network T^*, and we sampled sampled 10,000 data points from it (all variables are binary). From this data, we tried finding the best k'-hyperforest from the data. For $k' = k$, both algorithms were able to recover the hidden hypertree. For $k' > k$, MDL was able to recover the structure, while ML overfit the data with a model whose tree-width was k', whereas the true tree-width

[4] These variables are called the *Markov blanket*.

k	0	1	2	3	4	5
LL	-29.54	-16.89	-15.26	-14.99	-14.75	-14.48

Table 2. Test log-likelihood for ML-k-J on the ALARM network. The log-likelihood (negative entropy) of the ALARM network is -13.26. Numbers are averaged over 10 trials.

is k.

6.2. ALARM network

In another experiment, we generated a training set of 10,000 data points and a test set of 2,000 data points using the Bayesian network ALARM (Heckerman et al., 1995). Our goal was to approximate this network by using a bounded tree-width Markov network. In this case, ML found a better network than MDL. We evaluate the quality of a network by computing the log-likelihood on the held out test data. The test log-likelihood increased with the tree-width (Table 2).

6.3. MNIST handwritten digit recognition

The MNIST data set includes 28×28 4-bit grayscale pixels, which we downsampled to 14×14. We trained on the 60,000 examples and tested on the remaining 10,000. Due to time and space limits, we only tried tree-widths 1 and 2. ML-C_f-2 achieves 5.845% classification error, while Chow and Liu achieve 6.875% error. Here, having a separate model for each class (digit) helps because the relationship between pixels is different for each digit. Indeed, the structures of the Markov networks for each digit roughly outlines the digit.

6.4. UCI machine learning data sets

We tested our algorithm on 28 data sets from the UCI machine learning repository (Blake et al., 1998), summarized in Table 3.

In a preprocessing step, numeric variables that had at least 10 distinct values, were discretized into 5 intervals, with each interval containing the same number of points.

Although hyperforests have the ability to deal with data points with missing variables, we decided for simplicity to designate "missing" as an ordinary value that a variable can take on. We apply smoothing by adding a pseudo-count of 0.001 to each possible configuration for each hyperedge.

For each data set, we ran 50 trials, except connect4 (10 trials) due to time complexity. In each trial, we randomly split all the data points into a training set containing 90% of the points and a test set containing 10%. We trained each algorithm on the training set and measured 4 values: likelihood on the training set, classification error on the training set, likelihood on the test set, and classification error on the test set.

Data set	n	m	$\|Y\|$	r
adult	15 (6 N, 3 M)	48842	2	8.9
australian	15 (7 N, 0 M)	690	2	3.9
breast	10 (8 N, 1 M)	699	2	5.2
car	7 (0 N, 0 M)	1728	4	3.6
connect4	43 (0 N, 0 M)	67557	3	3.0
crx	16 (6 N, 7 M)	690	2	4.9
dna	181 (0 N, 0 M)	3186	3	2.0
ecoli	8 (5 N, 0 M)	336	8	4.6
flare	13 (0 N, 0 M)	1066	6	3.7
german	21 (3 N, 0 M)	1000	2	4.0
glass	10 (9 N, 0 M)	214	6	5.1
heart	14 (5 N, 0 M)	270	2	3.6
hepatitis	20 (6 N, 15 M)	155	2	3.6
ionosphere	35 (32 N, 0 M)	351	2	4.7
iris	5 (4 N, 0 M)	150	3	4.6
letter	17 (16 N, 0 M)	20000	26	6.2
lymphography	19 (0 N, 0 M)	148	4	3.3
mushroom	23 (0 N, 1 M)	8124	2	5.2
nursery	9 (0 N, 0 M)	12960	5	3.6
pima	9 (8 N, 0 M)	768	2	4.7
satimage	37 (36 N, 0 M)	6435	6	5.0
segment	20 (16 N, 0 M)	2310	7	4.8
shuttle	10 (9 N, 0 M)	58000	7	5.2
soybean	36 (0 N, 34 M)	683	19	4.2
splice	61 (0 N, 0 M)	3175	3	4.0
tic-tac-toe	10 (0 N, 0 M)	958	2	2.9
vote	17 (0 N, 16 M)	435	2	2.9
waveform	22 (21 N, 0 M)	5000	3	4.9
wine	14 (13 N, 0 M)	178	3	4.9

Table 3. Statistics about the data sets. n is the total number of variables. In addition, we give the number of numeric variables with at least 10 different values (N) and the number of variables with missing values (M). m is the number of data points, $|Y|$ is the number of classes, r is the average number of possible values per variable.

Tables 4 shows the average log-likelihood of the test set over all the data sets; Table 5 shows the classification error on the test set. In case of a tie, the most simple method is chosen.

To measure overall quality of each method, we averaged the measurements across all datasets (Table 6). On the training data, the most complex model family, ML-C_f-3, is clearly the best for both maximizing log-likelihood and minimizing classification error. However, it overfits the data, and performs badly on the test data.

On the test data, MDL generally performed better than ML. For maximizing the log-likelihood, C_f is better than J or C_s for every data set (fixing the other two method arguments), since C_f allows a more fine-grained modeling of class-specific dependencies. For minimizing classification error, C_s outperformed the other two representations. Although it can be useful to model each class separately, C_f suffers because the data spread very thin across all the classes. For den-

Data set	Best method	Best	MDL-C_f-3	CL	TAN
adult	MDL-C_s-2	-18.47	-18.47	-18.61	-18.61
australian	MDL-J-2	-19.60	-19.87	-20.30	-20.28
breast	MDL-C_s-1	-11.71	-11.71	-12.46	-12.13
car	MDL-C_s-1	-11.16	-11.18	-11.20	-11.16
connect4	ML-C_f-3	-21.34	-21.46	-22.51	-22.52
crx	ML-J-1	-21.85	-21.96	-23.42	-23.52
ecoli	MDL-J-2	-10.66	-11.26	-12.90	-12.16
flare	MDL-C_f-3	-8.91	-8.91	-9.37	-9.23
german	MDL-J-1	-29.61	-29.83	-30.35	-30.52
glass	MDL-J-1	-18.29	-19.54	-23.73	-22.46
heart	MDL-J-1	-19.98	-20.09	-22.43	-22.06
hepatitis	MDL-C_s-1	-24.72	-25.10	-31.87	-30.83
ionosphere	MDL-C_f-1	-57.00	-57.00	-58.72	-59.08
iris	MDL-J-1	-6.78	-7.14	-6.95	-6.95
letter	ML-C_f-3	-18.19	-19.41	-19.62	-20.55
lymph.	MDL-J-1	-22.16	-23.36	-27.44	-26.30
mushroom	ML-C_f-3	-13.14	-13.89	-16.02	-17.54
nursery	MDL-C_f-2	-13.91	-13.91	-13.94	-13.96
pima	ML-J-1	-15.53	-15.72	-15.91	-15.80
satimage	ML-C_f-2	-29.47	-30.81	-30.86	-31.74
segment	MDL-C_f-3	-19.47	-19.47	-19.51	-20.03
shuttle	ML-C_f-3	-6.19	-6.27	-6.72	-6.86
soybean	MDL-C_f-1	-18.25	-18.29	-21.50	-22.62
splice	ML-C_s-1	-114.30	-114.62	-114.49	-114.30
tic-tac-toe	ML-C_s-2	-13.73	-13.97	-14.40	-14.36
vote	ML-J-1	-15.28	-15.77	-15.78	-15.79
waveform	MDL-C_f-2	-38.66	-38.66	-38.80	-38.83
wine	MDL-J-1	-25.69	-26.24	-33.07	-32.29

Table 4. The test log-likelihood is given for the best method out of the ones we tried, MDL-C_f-3 (which had the overall best performance), Chow and Liu (CL), and Tree-Augmented Naive Bayes (TAN).

Data set	Best family	Best	MDL-C_s-3	CL	TAN
adult	MDL-J-3	0.15	0.15	0.15	0.15
australian	MDL-J-1	0.12	0.13	0.16	0.16
breast	MDL-C_s-3	0.02	0.02	0.03	0.03
car	ML-C_s-3	0.02	0.06	0.05	0.05
connect4	ML-C_s-3	0.19	0.19	0.24	0.24
crx	MDL-C_f-1	0.13	0.14	0.20	0.20
ecoli	MDL-C_s-3	0.15	0.15	0.22	0.22
flare	MDL-J-3	0.25	0.26	0.26	0.27
german	MDL-C_s-0	0.24	0.24	0.28	0.28
glass	MDL-C_f-3	0.33	0.36	0.35	0.41
heart	MDL-J-3	0.17	0.18	0.24	0.23
hepatitis	ML-J-1	0.15	0.15	0.16	0.18
ionosphere	MDL-C_f-1	0.08	0.09	0.09	0.09
iris	MDL-J-3	0.03	0.04	0.07	0.04
letter	ML-C_f-2	0.22	0.30	0.27	0.29
lymph.	MDL-C_s-3	0.18	0.18	0.23	0.36
mushroom	MDL-C_s-3	0.00	0.00	0.00	0.00
nursery	ML-C_s-3	0.02	0.06	0.05	0.06
pima	MDL-J-3	0.24	0.24	0.27	0.26
satimage	MDL-C_s-3	0.14	0.14	0.15	0.14
segment	ML-C_f-1	0.08	0.09	0.08	0.10
shuttle	MDL-C_s-3	0.02	0.02	0.02	0.02
soybean	MDL-C_f-1	0.05	0.07	0.06	0.13
splice	ML-J-2	0.03	0.05	0.05	0.04
tic-tac-toe	ML-C_s-3	0.08	0.16	0.26	0.23
vote	ML-J-1	0.04	0.07	0.07	0.07
waveform	MDL-C_s-3	0.19	0.19	0.19	0.19
wine	MDL-C_f-1	0.02	0.02	0.06	0.06

Table 5. The test classification error is given for the best method out of the ones we tried, MDL-C_s-3 (which had the overall best performance), Chow and Liu (CL), and Tree-Augmented Naive Bayes (TAN).

Method	train LL	train error	test LL	test error
ML-C_f-0 (NB)	-25.82	0.142	-26.44	0.159
ML-C_f-1 (CL)	-21.14	0.084	-24.75	0.153
ML-C_f-2	-18.91	0.051	-28.71	0.206
ML-C_f-3	**-16.54**	**0.036**	-30.72	0.295
ML-C_s-1 (TAN)	-21.56	0.089	-24.73	0.161
ML-C_s-2	-19.32	0.063	-28.19	0.243
ML-C_s-3	**-16.94**	**0.043**	-30.20	0.336
ML-J-1	-23.27	0.164	-24.48	0.177
ML-J-2	-20.88	0.130	-26.89	0.175
ML-J-3	**-18.41**	0.097	-29.92	0.206
MDL-J-1	-23.61	0.170	-24.37	0.182
MDL-J-2	-23.02	0.155	-23.82	0.168
MDL-J-3	-22.92	0.149	-23.73	0.163
MDL-C_s-1	-22.46	0.109	-23.76	0.139
MDL-C_s-2	-22.31	0.106	-23.62	**0.135**
MDL-C_s-3	-22.27	0.105	-23.58	**0.134**
MDL-C_f-1	-22.05	0.106	**-23.54**	0.139
MDL-C_f-2	-21.90	0.102	**-23.39**	0.136
MDL-C_f-3	-21.87	0.102	**-23.35**	0.137

Table 6. Performance for a few model families averaged over all data sets. The best figures are bolded.

sity estimation, the overall best method is MDL-C_f-3, which performed at least as well as TAN and CL on 25 out of the 28 datasets. For classification, the overall best method is MDL-C_s-3, which performed at least as well as TAN and CL on 21 out of the 28 datasets.

However, out of the 8 data sets containing at least 5000 examples, ML-C_f-2 performs better than MDL-C_f-3 in test log-likelihood (-20.21 versus -20.36), and MDL-C_f-3 performs better than MDL-C_s-3 (0.127 versus 0.132). With more data sets, we can afford to transition to more complicated models, from C_s to C_f and from MDL to ML.

The graphical model helps us understand the relationships in data. For example, on the splice data set, the variables form a linear sequence of DNA base pairs. It is expected that dependencies will be spatially local, and the class variable to have interactions mostly with the base pairs around the splice site. The hypertree structure of ML-J-3 reflects this.

7. Discussion

We introduced several methods by varying the tree-width, scoring metric, and model representation. We demonstrated how Markov networks with tree-width 2 and 3 can improve performance in both density estimation and classification. Increasing tree-width allows us to capture more dependencies between the variables, but can lead to overfitting with the ML scoring metric. MDL largely prevents this problem.

We could also consider Bayesian model averaging as it is applied in Meila and Jordan (2000; 2003) to trees. We can formulate a decomposable prior over hypertrees as for trees. To further improve density estimation and classification performance, we may wish to

model continuous variables with a Gaussian distribution (Friedman et al., 1998), and use inference to fill in the missing values of query points prior to classification.

As we increase tree-width in order to model the data more accurately, we pay a severe penalty. The time and space complexity of our algorithm increases exponentially with the tree-width. From m data points, we construct a complete k-hypergraph of all $O(n^{k+1})$ candidate hyperedges. A major bottleneck is constructing this hypergraph from the data, which requires $O(mn^{k+1})$ time. Pelleg and Moore (2002) suggests sampling a small portion of the training data to get an initial estimate of edge weights and sampling more data to refine the estimate as necessary. They showed that it drastically reduces the running time while sacrificing a little quality. Adapting this technique to hypertrees would bring bounded tree-width Markov networks closer to the practicality of Chow and Liu, while also allowing us to explore higher tree-width.

References

Bach, F., & Jordan, M. (2001). Thin junction trees. *Advances in Neural Information Processing Systems*.

Blake, C., Keogh, E., & Merz, C. (1998). Uci repository of machine learning databases.

Bouckaert, R. R. (1994). Minimum description length principle.

Cerquides, J., & de Mantaras, R. (2003). Tractable bayesian learning of tree augmented naive bayes classifiers.

Chickering, D. M., Geiger, D., & Heckerman, D. (1994). *Learning bayesian networks is np-hard* (Technical Report MSR-TR-94-17). Microsoft Research.

Chow, C. K., & Liu, C. N. (1968). Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory, IT-14*, 462–467.

Cormen, T. H., Leiserson, C. E., & Rivest, R. L. (1989). *Introduction to algorithms*. MIT Press.

Dasgupta, S. (1999). Learning polytrees. *Proceedings of the Conference on Uncertainty in Artificial Intelligence*.

Friedman, N., Geiger, D., & Goldszmidt, M. (1997). Bayesian network classifiers. *Machine Learning, 29*, 131–163.

Friedman, N., Goldszmidt, M., & Lee, T. J. (1998). Bayesian network classification with continuous attributes: getting the best of both discretization and parametric fitting. *Proceedings of the 15th International Conference on Machine Learning* (pp. 179–187). Morgan Kaufmann, San Francisco, CA.

Heckerman, D., Geiger, D., & Chickering, D. (1995). Learning bayesian networks: the combination of knowledge and statistical data. *20(3)*, 197–243.

Huang, K., King, I., & Lyu, M. (2002a). Constructing a large node chow-liu tree based on frequent itemsets. *Proceedings of the International Conference on Neural Information Processing*.

Huang, K., King, I., & Lyu, M. R. (2002b). Learning maximum likelihood semi-naive bayesian network classifier.

Liang, P., & Srebro, N. (2003). *A dynamic data structure for checking hyperacyclicity* (Technical Report). Massachusetts Institute of Technology.

Malvestuto, F. M. (1991). Approximating discrete probability distributions with decomposable models. *IEEE Transactions on Systems, Man and Cybernetics, 21*, 1287–1294.

Meila, M., & Jordan, M. (2000). Learning with mixtures of trees. *Journal of Machine Learning Research, 1*, 1–48.

Pearl, J. (1997). *Probabilistic reasoning in intelligent systems*. Morgan Kaufmann Publishers. Revised second printing edition.

Pelleg, D., & Moore, A. (2002). *Using tarjan's red rule for fast dependency tree construction* (Technical Report CMU-CS-02-116). Carnegie Mellon University.

Rissanen, J. (1987). Stochastic complexity. *J. Royal Statistical Society, Series B, 49*, 223–239.

Srebro, N. (2000). Maximum likelihood markov networks: An algorithmic approach. Master's thesis, Massachusetts Institute of Technology.